UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/730,900	12/10/2003	Jonathan Maron	19111.0117	5194

23517 7590 01/26/2007

BINGHAM MCCUTCHEN LLP
3000 K STREET, NW
BOX IP
WASHINGTON, DC 20007

EXAMINER	
CHEN, QING	
ART UNIT	**PAPER NUMBER**
2191	

SHORTENED STATUTORY PERIOD OF RESPONSE	MAIL DATE	DELIVERY MODE
3 MONTHS	01/26/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

If NO period for reply is specified above, the maximum statutory period will apply and will expire 6 MONTHS from the mailing date of this communication.

Office Action Summary	Application No.	Applicant(s)	
	10/730,900	MARON, JONATHAN	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *10 December 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-45* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-45* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *10 December 2003* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This is the initial Office action based on the application filed on December 10, 2003.

2. **Claims 1-45** are pending.

Drawings

3. The drawings are objected to as failing to comply with 37 CFR 1.84(p)(4) because:

- Reference character "102" has been used to designate both "client" and "browser" in Figure 1.

Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application.

The drawings are objected to because:

- The elements "browser" and "standalone" in Figure 1 are not described in the specification.

Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application.

Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief

description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either "Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not accepted by the Examiner, the Applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

Specification

4. The abstract of the disclosure is objected to because the first sentence is missing a verb. It should presumably read "A system and method for generating code that reduces the cost for large projects and that provides easier and quicker ways of finding errors and making modifications is disclosed." Correction is required. See MPEP § 608.01(b).

5. The disclosure is objected to because of the following informalities:

- The specification contains the following typographical errors:
  - The hyphen (-) between the words "markup" and "language" should be deleted on page 3, lines 11, 13, and 14, and page 4, lines 3 and 4.
  - The reference number "404" should be changed to "402" on page 16, line 5, since reference number "402" is used to designate "SAXParser" in Figure 4.
  - The reference number "402" should be changed to "404" on page 16, lines 5 and 8, since reference number "404" is used to designate "SAXReader" in Figure 4.
- The specification does not explain what the acronym EIS stands for.

Appropriate correction is required.

6. The use of trademarks, such as EJB and JAVA, has been noted in this application. Trademarks should be capitalized wherever they appear (capitalize each letter OR accompany each trademark with an appropriate designation symbol, *e.g.*, ™ or ®) and be accompanied by the generic terminology (use trademarks as adjectives modifying a descriptive noun, *e.g.*, "the JAVA programming language").

Although the use of trademarks is permissible in patent applications, the proprietary nature of the marks should be respected and every effort made to prevent their use in any manner, which might adversely affect their validity as trademarks.

Claim Objections

7. **Claims 5, 14, 20, 29, 31, 35, and 44** are objected to because of the following informalities:

- **Claims 5, 14, 20, 29, 35, and 44** contain a typographical error: a period (.) should be added at the end of the claim body.
- **Claim 31** contains the following typographical errors:
  - The word "and" should be added after the first limitation.
  - A colon (:) should be added after the phrase "the steps of" in the second limitation.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

8. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

9. **Claims 1-15, 18-30, 33-37, and 42-45** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 1 recites the limitation "creating an event handler for method nodes." It is noted that Claim 5, which depends on Claim 1, recites "creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node." Thus, the claim is rendered indefinite, since there is no congruence between the recited limitations in the two claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "creating an event handler for a method node" for the purpose of further examination.

Claims 2-15 depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.

Claim 2 recites the limitation "the class file." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the archive file" for the purpose of further examination.

Consequently, the limitation will read "wherein the archive file is an Enterprise Java Bean archive file."

Claims 3-6 depend on Claim 2 and, therefore, suffer the same deficiency as Claim 2.

Claims 5, 6, 14, 15, 20-22, 24, 29, 30, 35, 36, 44, and 45 recite the limitation "the markup language description." There is proper explicit antecedent basis for the limitation, however, the claims are rendered indefinite because it is unclear to the Examiner whether "the markup language description" is referring to "Extensible Markup Language description" or not. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the markup language description" for the purpose of further examination.

Claim 23 depends on Claim 22 and, therefore, suffers the same deficiency as Claim 22.

Claims 25-28 depend on Claim 24 and, therefore, suffer the same deficiency as Claim 24.

Claims 6, 21, 36, and 45 recite the limitation "the registered Simple Application Programming Interface for Extensible Markup Language event handler." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the Simple Application Programming Interface for Extensible Markup Language event handler" for the purpose of further examination.

Claims 22-30 depend on Claim 21 and, therefore, suffer the same deficiency as Claim 21.

Claims 15, 30, and 45 recite the limitation "the plurality of registered Simple Application Programming Interface for Extensible Markup Language event handlers." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers" for the purpose of further examination.

Claims 18 and 27 recite the limitations "systems included in the input class" and "parameters of the system." It is unclear to the Examiner what the systems are referring to and how systems can be included in an object-oriented class, which only defines methods and properties for an object instance. Furthermore, the specification only discloses "extracting information relating to the objects, methods, etc., that are included in the class. For example, information identifying methods included in the input class may be extracted and for each method, information relating to parameters of the method are extracted" *(see Page 8: 18-19 through Page 9: 1-3)*. In the interest of compact prosecution, the Examiner subsequently interprets these limitations as reading "methods included in the input class" and "parameters of the method," respectively, for the purpose of further examination. Consequently, the latter

limitation will read "for each method, extracting information relating to parameters of the method."

Claims 33 and 42 recite the limitations "computer program products included in the input class" and "parameters of the computer program product." These claims are rejected for the same reasons set forth in the rejections of Claims 18 and 27.

Claims 19-26 depend on Claim 18 and, therefore, suffer the same deficiency as Claim 18.

Claims 28-30 depend on Claim 27 and, therefore, suffer the same deficiency as Claim 27.

Claims 34-36 depend on Claim 34 and, therefore, suffer the same deficiency as Claim 34.

Claims 43-45 depend on Claim 42 and, therefore, suffer the same deficiency as Claim 42.

Claims 20, 22, and 29 recite the limitation "a system node found in the markup language description." However, the parent claim of Claims 20, 22, and 29 (Claim 16) recites the limitation of "a method node found in the markup language description." Therefore, it is unclear to the Examiner how the system node is related to the method node. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "a method node found in the markup language description" for the purpose of further examination.

Claims 35, 37, and 44 recite the limitation "a computer program product node found in the markup language description." These claims are rejected for the same reasons set forth in the rejections of Claims 20, 22, and 29.

Claim 21 depends on Claim 20 and, therefore, suffers the same deficiency as Claim 20.

Claims 23-28 depend on Claim 22 and, therefore, suffer the same deficiency as Claim 22.

Claim 30 depends on Claim 29 and, therefore, suffers the same deficiency as Claim 29.

Claim Rejections - 35 USC § 101

10. 35 U.S.C. 101 reads as follows:

> Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

11. **Claims 31-45** are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 31-45 recite computer readable medium as a claimed element. However, it is noted that the specification describes computer readable media to include transmission-type media, such as digital and analog communication links *(see Page 24: 1-2)*. Consequently, the computer readable medium can be reasonably interpreted as carrying electrical signals.

Claims that recite nothing but the physical characteristics of a form of energy, such as a frequency, voltage, or the strength of a magnetic field, define energy or magnetism *per se*, and as

such are non-statutory natural phenomena. *O'Reilly v. Morse*, 56 U.S. (15 How.) 62, 112-14 (1853). Moreover, it does not appear that a claim reciting a signal encoded with functional descriptive material falls within any of the categories of patentable subject matter set forth in § 101.

Claim Rejections - 35 USC § 103

12. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

13. **Claims 1-45** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Golden (US 6,925,631)** in view of **Sarkar et al. (US 6,754,659)**.

As per **Claim 1**, Golden discloses:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.")*;

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the*

org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.");

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.")*.

However, Golden does not disclose:

- receiving an archive file to be deployed; and

- introspecting an input class included in the archive file to generate information relating to the input class.

Sarkar et al. disclose:

- receiving an archive file to be deployed *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.")*; and

- introspecting an input class included in the archive file to generate information relating to the input class *(see Figure 4: 400; Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include receiving an archive file to be deployed; and introspecting an input class included in the archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13)*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would

be obvious because one of ordinary skill in the art would be motivated to utilize an industry standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61).*

As per **Claim 3**, the rejection of **Claim 2** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 5**, the rejection of **Claim 4** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 6**, the rejection of **Claim 5** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found,*

an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 7**, the rejection of **Claim 1** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 8**, the rejection of **Claim 7** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.").*

As per **Claim 9**, the rejection of **Claim 8** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined*

by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 11**, the rejection of **Claim 10** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would be obvious because one of ordinary skill in the art would be motivated to utilize an industry

standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61)*.

As per **Claim 12**, the rejection of **Claim 11** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 13**, the rejection of **Claim 12** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 15**, the rejection of **Claim 14** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an*

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 16**, Golden discloses:

- a processor operable to execute computer program instructions *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*;

- a memory operable to store computer program instructions executable by the processor *(see Column 7: 34-35, "... a computer system 10 with a processing unit and storage 11 for processing programs.")*; and

- computer program instructions stored in the memory and executable *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* to perform the steps of:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.")*;

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing*

process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.");

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.")*.

However, Golden does not disclose:

- receiving an archive file to be deployed; and

- introspecting an input class included in the archive file to generate information relating to the input class.

Sarkar et al. disclose:

- receiving an archive file to be deployed *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.")*; and

- introspecting an input class included in the archive file to generate information relating to the input class *(see Figure 4: 400; Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include receiving an archive file to be deployed; and introspecting an input class included in the archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 17**, the rejection of **Claim 16** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would

be obvious because one of ordinary skill in the art would be motivated to utilize an industry standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61).*

As per **Claim 18**, the rejection of **Claim 17** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13)*.

As per **Claim 19**, the rejection of **Claim 18** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.")*.

As per **Claim 20**, the rejection of **Claim 19** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*.

As per **Claim 21**, the rejection of **Claim 20** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found,*

an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.").*

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined*

by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 25**, the rejection of **Claim 24** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 26**, the rejection of **Claim 25** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would be obvious because one of ordinary skill in the art would be motivated to utilize an industry

standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61).*

As per **Claim 27**, the rejection of **Claim 26** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 28**, the rejection of **Claim 27** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 29**, the rejection of **Claim 28** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 30**, the rejection of **Claim 29** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an*

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 31**, Golden discloses:

- a computer readable medium *(see Column 6: 25-28, "... a computer-readable data carrier ...")*; and

- computer program instructions, recorded on the computer readable medium, executable by a processor, *(see Column 6: 25-28, "The disclosed embodiments of the computer program product comprise the disclosed program code which, for example, is stored on a computer-readable data carrier ...")* for performing the steps of:

- generating a markup language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.")*;

- creating an event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.")*;

- registering the event handler *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.")*;

- parsing the markup language description and invoking the registered event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.")*; and

- generating output code using the invoked event handler *(see Column 14: 45-46, "... the taglet document is written to the output stream 15.")*.

However, Golden does not disclose:

- receiving an archive file to be deployed; and

- introspecting an input class included in the archive file to generate information relating to the input class.

Sarkar et al. disclose:

- receiving an archive file to be deployed *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.")*; and

- introspecting an input class included in the archive file to generate information relating to the input class *(see Figure 4: 400; Column 6: 1-10, "... a support code generation*

tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include receiving an archive file to be deployed; and introspecting an input class included in the archive file to generate information relating to the input class. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 32**, the rejection of **Claim 31** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would be obvious because one of ordinary skill in the art would be motivated to utilize an industry

standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61).*

As per **Claim 33**, the rejection of **Claim 32** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 34**, the rejection of **Claim 33** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 35**, the rejection of **Claim 34** is incorporated; and Golden further discloses:

- creating a Simple Application Programming Interface for Extensible Markup Language event handler for a method node found in the markup language description *(see Column 9: 37-41, "The SAX parser, an event-driven API, is used for the parsing process. The application registers an event handler to a parser object that implements the org.sax.Parser interface. The event handler interface DocumentHandler is called whenever an element is found in the input stream.").*

As per **Claim 36**, the rejection of **Claim 35** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the Simple Application Programming Interface for Extensible Markup Language event handler *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found,*

an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked."):

As per **Claim 37**, the rejection of **Claim 31** is incorporated; and Golden further discloses:

- creating a plurality of event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 38**, the rejection of **Claim 37** is incorporated; and Golden further discloses:

- registering each of the plurality of event handlers *(see Column 9: 38-40, "The application registers an event handler to a parser object that implements the org.sax.Parser interface.").*

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and Golden further discloses:

- parsing the markup language description and invoking each of the plurality of registered event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an XML tag is found, an object (e.g. a JAVA class), as defined*

by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

As per **Claim 40**, the rejection of **Claim 39** is incorporated; and Golden further discloses:

- generating output code using each of the plurality of invoked event handler in parallel *(see Column 14: 45-46, "... the taglet document is written to the output stream 15."; Column 18: 12-13, "If the branching is not conditional, the two branches following the first engine work in parallel.").*

As per **Claim 41**, the rejection of **Claim 40** is incorporated; however, Golden does not disclose:

- wherein the archive file is an Enterprise Java Bean archive file.

Sarkar et al. disclose:

- wherein the archive file is an Enterprise Java Bean archive file *(see Column 5: 64-67, "This single generic EJB is then deployed in an EJB container in a known manner, where it is available to run the original Java beans in an EJB environment.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include wherein the archive file is an Enterprise Java Bean archive file. The modification would be obvious because one of ordinary skill in the art would be motivated to utilize an industry

standard architecture for running server-side business logic, providing additional benefits of locatability (sic) in a network, and scalability *(see Sarkar et al. – Column 2: 56-61).*

As per **Claim 42**, the rejection of **Claim 41** is incorporated; however, Golden does not disclose:

- extracting information identifying methods included in the input class; and
- for each method, extracting information relating to parameters of the method.

Sarkar et al. disclose:

- extracting information identifying methods included in the input class *(see Column 6: 1-10, "... a support code generation tool 504, utilizing Java's introspection capability determines the setter/getter methods and execution methods of original Java bean 502 to be emulated.")*; and
- for each method, extracting information relating to parameters of the method *(see Column 6: 11-14, "Based also on the signature of original Java bean 502, a properties class 508 is generated (step 404) which is configured to contain input and output properties matching those of original Java bean 502.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Sarkar et al. into the teaching of Golden to include extracting information identifying methods included in the input class; and for each method, extracting information relating to parameters of the method. The modification would be obvious because one of ordinary skill in the art would be motivated to extract information from

applications without modifying the applications themselves *(see Sarkar et al. – Column 2: 10-13).*

As per **Claim 43**, the rejection of **Claim 42** is incorporated; and Golden further discloses:

- generating an Extensible Markup Language description of the input class based on the generated information relating to the input class *(see Figure 1; Column 6: 36-38, "FIG. 1 shows an example of an extensible markup language input stream, here a fragment of a XML document which represents a small section of an order.").*

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Golden further discloses:

- creating a plurality of Simple Application Programming Interface for Extensible Markup Language event handlers for a method node found in the markup language description *(see Figure 2; Column 6: 51-61, "FIG. 2 illustrates that the tags of the XML fragment of FIG. 1 are mapped to classes of an object-oriented programming language by arrows pointing from each tag to a class. More specifically, the arrows point to "init" methods at the start-tags and to "run" methods at the end-tags.").*

As per **Claim 45**, the rejection of **Claim 44** is incorporated; and Golden further discloses:

- parsing the markup language description using a Simple Application Programming Interface for Extensible Markup Language parser and invoking the plurality of Simple Application Programming Interface for Extensible Markup Language event handlers *(see Column 9: 53-61, "Upon parsing the input stream, a DOM representation of it is created. As an*

XML tag is found, an object (e.g. a JAVA class), as defined by the corresponding binding, is bound to the DOM tree for the specific tag. A tag with behavior (e.g. a JAVA class) bound to it is called a "taglet". As will be explained in more detail below, a init ( ) method is invoked on the taglet. After all of the taglet's children (possibly zero) have been added to the DOM representation, the taglet's run ( ) method is invoked.").

Conclusion

14. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

A. **Skufca et al. (US 6,922,695)** disclose systems for accessing back-end enterprise applications and data from the Internet by providing an intelligent, real-time data and logic cache that is synchronized with back-office systems.

B. **Carroll, Jr. (US 6,990,654)** discloses a system and a method for utilizing an interface library to create application interfaces.

C. **Gerken (US 7,000,218)** discloses a system and method for creating JSP custom tag files in response to developer requests.

D. **Sharma et al. (US 7,159,224)** disclose methods, systems, and articles of manufacture for providing a servlet container based web service endpoint in a remote procedure call based distributed computing system.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The

Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

WEI ZHEN
SUPERVISORY PATENT EXAMINER

QC / QC
January 9, 2007